

**Identifying text phrases containing similar semantic meaning with search keywords using NLP**

**Y. T. L Somarathna**

**Index No : 16001402**

**Supervisor: Dr M. G. N. A. S. Fernando**

**<****February 2021>**

Submitted in partial fulfillment of the requirements of the

B.Sc in Computer Science Final Year Project (SCS4124)



# Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name: Y. T. L. Somarathna

………………………………………………

Signature of Candidate Date:

This is to certify that this dissertation is based on the work of

Dr. M. G. N. A. S. Fernando

under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Principle/Co- Supervisor’s Name: Mr W.V Welgama

………………………………………………

Signature of Supervisor Date:

# Abstract

A core strategy of content SEO is placing the right amount of keywords in a webpage to archive the optimal keyword density with related quality keywords. The current approach for performing this task is manually going through the content and checking for text phrases that can be replaced by keywords and replacing them. In this study, we focus on a specific domain of websites (Android) and try to develop a model that can automatically recognize text phrases that are semantically similar to keywords. Distributional semantics is used to capture the meaning of words and an un-supervised classifier is used to classify phrases related to a particular keyword with the help of words semantics captured by distributional semantics.

Even with the latest NLP advancement capturing and comparing the semantics of text phrases seems to be some problems as text phrases have less context to work with. This study shows that still, we can produce useful results in recognizing text phrases with similar semantic meaning with search keywords.

# Preface

This dissertation has been written for the partial fulfilment of the requirements of the B.Sc. in Computer Science (Hons) Final Year Project in Computer Science (SCS4124). I was engaged in this research and writing this dissertation from April 2020 to February 2021.

This research work presents the studies of applying and deep learning techniques to identify word phrases that share a similar meaning with search keywords. To the best of my knowledge, research work on automating the keyword process of Search Engine Optimization using a deep learning approach or any other approach has not been carried out so far. Therefore, this work will be the first of it’s kind. We have created our dataset containing a large number of pairs of keywords and word phrases with similar meaning in the android domain for this study. We created a model using existing deep neural architecture BERT for classifying word phrases to the relevant keywords. Further, we did several experiments with this model and applied various steps to fine-tune the models to enhance the output. The result presented in Section 5 relies upon experiments conducted by me.

With the constant guidance and supervision of my supervisor and co-supervisor, conclusions were drawn on evaluation and training the models. This piece of research work would be a great source of knowledge for future research on using NLP in SEO.

# Acknowledgement

First, I would like to thank my university, the University of Colombo School of Computing (UCSC) for giving me this great opportunity to carry out individual research in which I could develop my research and other academic skills.

I would like to express my sincere gratitude to my research supervisor Dr M. G. N. A. S. Fernando and my co-supervisor Mr W.V Welgama for providing me with their valuable guidance and supervision throughout this research project. I am grateful to them for finding time to help me all the time, from the beginning to the end of this research project.

Further, I would like to pay my sincere gratitude to Dr H.N.D. Thilini, computer science project coordinator, and all other UCSC staffs members for all the assistance they provided to make this project successful.

# Table of Contents

[Declaration ii](#__RefHeading___Toc724_2832949180)

[Abstract iii](#__RefHeading___Toc726_2832949180)

[Preface iv](#__RefHeading___Toc728_2832949180)

[Acknowledgement v](#__RefHeading___Toc730_2832949180)

[Table of Contents vi](#__RefHeading___Toc732_2832949180)

[List of Figures viii](#__RefHeading___Toc734_2832949180)

[List of Tables ix](#__RefHeading___Toc736_2832949180)

[List of Acronyms x](#__RefHeading___Toc738_2832949180)

[Chapter 1 - Introduction 1](#__RefHeading___Toc740_2832949180)

[1.1 Background to the Research 1](#__RefHeading___Toc742_2832949180)

[1.2 Research Problem and Research Questions 2](#__RefHeading___Toc744_2832949180)

[1.3 Justification for the research 3](#__RefHeading___Toc746_2832949180)

[1.4 Methodology 4](#__RefHeading___Toc748_2832949180)

[1.5 Outline of the Dissertation 6](#__RefHeading___Toc750_2832949180)

[1.6 Delimitations of Scope 6](#__RefHeading___Toc754_2832949180)

[1.7 Conclusion 7](#__RefHeading___Toc756_2832949180)

[Chapter 2 - Literature Review 8](#__RefHeading___Toc3821_1516130272)

[2.1 Why is placing keywords on Content important? 8](#__RefHeading___Toc3823_1516130272)

[2.2 How to place keywords and gain a better search rank? 9](#__RefHeading___Toc3825_1516130272)

[2.3 How to identify keyword replaceable word phrases via Semantic Matching in NLP? 10](#__RefHeading___Toc3827_1516130272)

[2.4 Distributional Semantics for short text similarity 11](#__RefHeading___Toc3829_1516130272)

[2.5 Problem with the traditional word embeddings and the solution 14](#__RefHeading___Toc1545_721462700)

[Chapter 3 - Design 17](#__RefHeading___Toc760_2832949180)

[3.1 Data Collection and Preprocessing 19](#__RefHeading___Toc1547_721462700)

[3.2 Model Implementation and Keyword Suggestion generation 21](#__RefHeading___Toc1549_721462700)

[3.3 Evaluation 24](#__RefHeading___Toc1551_721462700)

[Chapter 4 - Implementation 25](#__RefHeading___Toc762_2832949180)

[4.1 Dataset creation and preprocessing 25](#__RefHeading___Toc1954_721462700)

[4.2 Model Implementation 25](#__RefHeading___Toc1956_721462700)

[Chapter 5 - Results and Evaluation 30](#__RefHeading___Toc764_2832949180)

[Chapter 6 - Conclusions 32](#__RefHeading___Toc766_2832949180)

[6.1 Introduction 32](#__RefHeading___Toc768_2832949180)

[6.2 Conclusions about research questions (aims/objectives) 32](#__RefHeading___Toc770_2832949180)

[6.3 Conclusions about research problem 34](#__RefHeading___Toc772_2832949180)

[6.4 Limitations 34](#__RefHeading___Toc774_2832949180)

[6.5 Implications for further research 35](#__RefHeading___Toc776_2832949180)

[References 36](#__RefHeading___Toc778_2832949180)

[Appendix A: Code Listings 38](#__RefHeading___Toc780_2832949180)

[A.1 Code snippet of Dataset creation and Preprocessing 38](#__RefHeading___Toc784_2832949180)

[A.2 Code snippet of semantic encoding of text phrases 42](#__RefHeading___Toc2109_721462700)

[A.3 Code snippet of custom K-Means Implementation 42](#__RefHeading___Toc2111_721462700)

[A.4 Code snippet for training the K-Means model 53](#__RefHeading___Toc2113_721462700)

[A.5 Code snippet for evaluating the model 54](#__RefHeading___Toc2115_721462700)

# List of Figures

[Figure 1: Hypothetical example — two-dimensional representation of the word embeddings for two short texts 13](#Figure!0|sequence)

[Figure 2: High-level architecture of the research design 18](#Figure!1|sequence)

[Figure 3: Screenshot of exported keywords from Google Keyword Planner 20](#Figure!2|sequence)

[Figure 4: Training Process of K-Means model 22](#Figure!3|sequence)

[Figure 5: Selecting the Winning Centroid 23](#Figure!4|sequence)

[Figure 6: Selecting the best matching keyword from winning centroid 24](#Figure!5|sequence)

[Figure 7: Equation for calculating the total time for suggesting a keyword for a text phrase 27](#Figure!6|sequence)

[Figure 8: Equation for number of centroids 28](#Figure!7|sequence)

# List of Tables

[Table 1: Results of evaluating the model 31](#Table!0|sequence)

# List of Acronyms

SEO Search Engine Optimization

NLP Natural Language Processing

Webmaster Web Content Creator

LCS Longest Common Substring

BOW Bag-of-words

LSA Latent Semantic Analysis

SVD Singular Value Decomposition

IDF Inverse Document Frequency

LSTM Long Short Term Memory

ETM Embedding based Topic Model

DSSM Deep Structured Semantic Models

BERT Bidirectional Encoder Representation from Transformers

# Introduction

## 1.1 Background to the Research

The web is changing rapidly both the content and the technologies used, So does the users and the search engines, which facilitates the users. Search engines use complex algorithms to determine the relevance of a certain document related to a particular search query and displays the relevant documents ranked by their importance. Those algorithms consider several factors to determine the importance of each document such as the keywords used within the content and the number of backlinks a particular website has. And those factors are constantly changing and evolving throughout the last decade. And the webmasters( Web Content Creators) are using various Search Engine Optimization(SEO) techniques to evolve their content to meet new standards of search engines to reach their audience. Studies have been carried out to determine the importance of SEO as a whole and also to determine the importance of various SEO strategies.

In [1] states that the ultimate SEO goal is to provide the basic policy to optimize websites, in order for the latter to succeed in higher and better-related rankings in the search engines, as well as better targeted trafﬁc, both in volume and depth.

The ratio between words in all the keywords of a webpage and all the words within the webpage is called the keyword density of a web page. Properly placing keywords within the content to archive the optimal keyword density and using the right set of keywords in that process is an essential part of any webmasters SEO strategy. As the current approach is done manually by an expert in SEO by analyzing keywords related to the niche of the website, and changing the content accordingly.

With the advancements of NLP, this study is focused on the first and major step of automating this process, which is building a system that can reliably identify text phrases that share similar semantic meaning with a search keyword.

## 1.2 Research Problem and Research Questions

Becoming skilled at SEO or hiring an expert on SEO has become an essential thing for any content creator or marketer on the internet to reach potential organic viewers, which can be very costly and time-consuming. So having a tool that can automatically optimize content for its relevant search metrics can benefit both the content creator and the reader at the same time as saves time and cost for SEO optimizations and also brings more relevant content for the reader. This study does not address the whole SEO optimization process.

But containing the proper keyword cluster in the optimal density within the web pages is the main limiting factor of the content creation as it takes a lot of time and effort to complete. Even though there are many other important aspects of SEO such as backlink creation and implementing search engine friendly architecture within the website, all those major SEO practices can be separated from the content creation and executed independently.

This research project intends to analyze the applicability of semantic matching based on distributional semantic NLP techniques to identify word phrases that can be replaced by word phrases containing keywords. As the first step to automate the process of optimizing the keyword density of web pages.

To address the requirement above, the proposed study will answer the following questions

* How to identify word phrases that contain a similar meaning with keywords related to content?
  + To identify a text phrase that shares a meaning with a search keyword we must have a way of measuring the semantic similarity between those two test phases.
* What are the techniques available to compare the similarity of texts?
  + Measuring the semantic similarity of texts( semantic matching) is categorized under NLP and has a lot of approaches for archiving this, which are based on various underlying techniques which are used appropriately as they suite the desired end result going to be archived by the semantic matching task.
* Which of those are applicable to compare semantic similarity between small word phrases?
  + Most of the semantic matching techniques are not suitable for short text similarity tasks especially the techniques which are not based on lexical matching, as they are not able to capture the semantic similarity over a trivial level.
* How to develop a model that can identify similarities between short text phrases and search keywords from those applicable techniques?
  + By combining the power of distributed semantics to capture the semantics of words and encoding them into text embeddings using an appropriate encoding technology and using an additional layer to classify text into relevant keywords.

## 1.3 Justification for the research

With the advancement of Artificial intelligence and machine learning, computers are becoming good at tasks that were seemingly impossible to perform by a computer. Which makes machine learning the go-to solution for most modern computer science problems.

This study explores the applicability of NLP techniques to enhance the keyword density of webpages and increase the overall SEO ranking of the webpages which reduces the time and effort needed to reach their full potential audience. Which saves them time and effort to create better and more content.

Human-error is a significant factor in SEO where final results highly depend on the expertise and the experience of the person who performs the SEO optimizations. So an accurate software solution that can perform this can omit that human-error.

Results of this study can enable and inspire studies on the inapplicability of NLP and machine learning in other areas of content SEO. Especially with the ability to understand the semantics of short word phrases related to the keywords, It can be used to filter out useful comments which actually add value for the content from spammy or unuseful comments, and also for automating the creation of internal links within the content.

And semantic matching over short text phrases seems to be attracting less attention throughout the literature. So the results of this study can also contribute to widening the understanding of semantic matching as it explores it in a not widely used context.

## 1.4 Methodology

As the first step, sufficient datasets for training the model and evaluating the model must be created. However, expanding this study over every niche of websites is not practical with this study. So websites of the niche “Android” have been chosen to carry out the research because it meets the following challenges of the study can be met with the “Android” niche.

* Need a domain understanding in creating datasets
* Has a large amount of content and a large audience that consumes the content
* Vocabulary used in the field changes rapidly

A dataset of keywords about the Android niche is created using keywords extracted from top websites about Android using Adword Keyword Planner[2]. And those keywords will be combined with word phrases, with the help of domain knowledge about “Android” to manually create the dataset consisting of keywords and word phrases with similar semantic meaning with keywords to evaluate the implemented model.

The proposed solution consists of two parts. Word embeddings, which is obtained by unsupervised learning and a classifier to classify word phrases to the keyword it belongs to. Both keywords and phrases to be matched needed to be encoded to a mathematical representation of the text phrases using the most suited mechanism described in the literature, in order to be classified.

According to the literature review carried out, the BERT model is one of the best achievements of modern deep learning-based NLP, which produce the state of the art solutions for a lot of problems of NLP that has the best ability to capture the semantics. So I decided to use a pre-trained BERT model to capture the semantic of word phrases.

So In this study, a pre-trained BERT model is used to capture the semantics of keywords and candidate word phrases to classify as share the semantic meaning with a particular keyword as vectors of semantics. Further details about this can be found in chapter 3. So with these semantic vectors, we can use a distance measure such as euclidian distance to match the best semantically matching keyword or top matching keywords for a given phrase. But here another problem arise, that is the number of keywords for a given niche is huge so holding the semantic vectors of the entire list of keywords is memory intensive and not practical, and embedding each keyword at the time of getting the distance measure is so time-consuming, so we have to optimize the problem on memory and time.

As a solution for this, a custom implemented k-means algorithm is used to cluster the keyword set into several centroids only the vectors of centroids were kept in the memory. So as the first step of the classification a winning centroid will be calculated and then all the keywords in that particular centroid will be encoded with the BERT model and best matching keywords can be found from here by applying a distance measure with those keywords semantic vectors.

## 1.5 Outline of the Dissertation

This thesis is organized as follows, In Chapter 2, a comprehensive study about existing techniques and approaches related to the domain of NLP and SEO is presented. The research design, along with the high-level architecture for addressing their search question, is presented in Chapter 3. Chapter 4 demonstrate the implementation details of the proposed methodology. In Chapter 5, comprehensive details of experiments carried out and evaluation result of all the model implemented is presented. Last chapter, Chapter 6 demonstrate the conclusion of the research and outlines future work.

## 1.6 Delimitations of Scope

In Scope

* Analysing the applicability of currently available distributed semantic techniques on capturing semantic similarity between phrases to find phrases that can be replaceable by phrases with keywords.
* Building a machine learning model which is capable of enhancing the keyword density of web pages by identifying word phrases that are replaceable by keywords.
* Training the above-implemented model and evaluating the results.

Out of Scope

* This study is only applicable to content made in English and does not intend to support multilingual content.
* This study does not focus on the other aspects of SEO such as backlink building and neither on improving the content quality or user experience.

## 1.7 Conclusion

This chapter laid the foundations for the dissertation. It introduced the research problem, research questions, research aim and objectives. Then the research was justified, the methodology was briefly described and justified, the dissertation was outlined, and the limitations of the research were given. Based on these foundations, the dissertation can proceed with a detailed description of the research and its contributions.

# Literature Review

## 2.1 Why is placing keywords on Content important?

In the context of SEO, a keyword is referred to as a word phrase which consists of a few words, which are bound for a specific subject. As an example, finding the keyword “Google play store” within the content of a website is an indication for a search engine to classify and rank that content under the “Android” website category.

Every search engine publishes a guideline for webmasters( Content Creators) to follow to achieve a higher ranking in their search engines. According to [1] and it clearly states on the webmaster’s guides of Google [3] and Bing [4] as having properly placed keywords in the text and tags of the webpage as a factor which is considered for calculating the search engine ranking of a webpage.

And effective keyword use and keyword research, also identified as a White Hat SEO technique [5], Which is not going to be penalised by search engines. And properly placed keywords helps search engines to better understand the website and the web content.

Placing keywords within the content is very effective in archiving better search engine rankings, As stated in [6] A website with 100 visitors a day can increase its website traffic from its current baseline of 100 to 200 visitors a day, and increase its conversion rate from 1% to 2%  within 30 days by using proper keywords in their content. Content creators can even use keywords that users commonly misspelt to direct traffic.

In [7] researchers create a supervised neural network model that estimates the position of a web resource based on the keywords within the web resource. They create a dataset consisting of 7400 data by googling search phrases related to three fields of Medicine, Student and Sport and collecting data according to 45 characteristics related to Number, Density and Prominence of keywords and split the dataset as 6000 to 1400 to train and test. At the evaluation, their model shows an accuracy of 89.36% with their testing dataset.

## 2.2 How to place keywords and gain a better search rank?

In [8] SEO strategy, according to the keyword optimization, the authors suggest increasing the keyword density by analysing the content using a keyword analysis tool and modifying the content as it contains the prettiest compactness in the ratio between the keywords word count and webpage word count.

Keyword density is a measure of the number of keywords word count relative to the total number of words in a webpage. To increase the keyword density in the content without losing the quality of the content the webmasters have to look for word phrases that share a similar meaning with search keywords and replace them with keywords.

So stated above the current approach for keyword optimization is using keyword analysis tools such as the SEO content analysis tool [9] and finding relevant keywords regards to the content and modifying content within the webpage as it contains more keywords, which is manually done by the content creator by finding word phrases which are semantically similar to the keywords and replacing them until he archives the desired keyword density.

## 2.3 How to identify keyword replaceable word phrases via Semantic Matching in NLP?

As the first step to automate this task, a system that can identify the keywords that are semantically similar to a word or a word phrase that is existing on the document is needed. If we input all the word phrases within a web page this system must be able to identify If there is any word phrase which possesses a meaning similar to the meaning expressed by a related keyword within the category of the website and also what is that category. In this study, we can use existing NLP technologies combined to archive this problem.

According to [10] many NLP  approaches have been suggested, based on lexical matching, handcrafted patterns, syntactic parse trees, external sources of structured semantic knowledge and distributional semantics. However, lexical features, like string matching, do not capture semantic similarity beyond a trivial level. Furthermore, handcrafted patterns and external sources of structured semantic knowledge cannot be assumed to be available in all circumstances and for all domains. And syntactic parse trees expect a well-structured sentence.

And also within short text phrases, they are less likely to find any structure or lexical features within phrases. And it is also finding keyword replaceable phrases by lexical similarities is almost pointless, as lexically similar phrases with a keyword are most probable for themselves to be keywords which are already existing in the content. Lastly, approaches depending on parse trees are restricted to syntactically well-formed texts, typically of one sentence in the length.

String-based similarity techniques such as the Longest Common Substring(LCS) and N-grams works by matching the lexical similarity between two terms. LCS calculates the longest common substring of two terms and the N-grams algorithm considers a subsequence of n items of the term, distance in N-Gram is computed by dividing the number of similar n-grams by the maximal number of n-grams available [11]. These approaches are especially challenging within short text similarity as the length of the strings to be compared plays a crucial role in these algorithms.

Bag-of-words(BOW) and Bag-of-n-grams approaches are very popular in most semantic matching problems because of their simplicity and efficiency. However, they are not suitable for short text similarity tasks as they have the following weaknesses as described in [12].

* have very little sense about the semantics of the words or more formally the distances between the words. This means that the words “powerful,” “strong” and “Paris” are equally distant despite the fact that semantically, “powerful” should be closer to “strong” than “Paris.”
* The word order is lost, and thus different sentences can have exactly the same representation, as long as the same words are used.

So in this study to be able to match the semantic similarity I need an approach where the semantics of each individual word has been preserved within their representations.

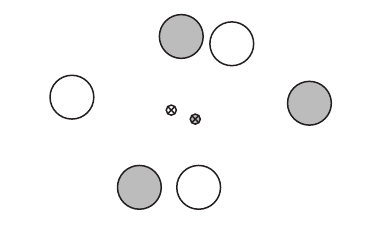
## 2.4 Distributional Semantics for short text similarity

However, techniques like distributional semantic which only depends on the semantic of each word can be corporated in this task. Distributed semantics lay on the idea meaning of a word can be described by its context, and with a large dataset semantics of the words can be approximated.

Especially in neural network-based algorithms like word2vec [13], and algorithms like Latent Semantic Analysis(LSA) [14] which based the full co-occurrence matrix and shrinking the matrix using singular value decomposition(SVD) and Glove [15] which can be identified as a hybrid of word2vec and LSA can train over a large amount of unlabeled data. And data will create a semantic space with each word containing a vector containing semantic values according to each dimension of the semantic space. And those vectors are called word embeddings.

Even though word embeddings are great for measuring the semantic distance between words, it has its own challenges if it is directly applied to short text comparison tasks. First of all, as it is applied at the word-level we need to come up with some mechanism to combine the meaning of each word within a text and create embeddings for text phrases to compute their semantics.  Even after that still, there are challenges such as each word doesn’t contribute to the meaning of the text equally and the combined position of phrases end up nearby in the semantic space, even though the individual words, word embeddings sit away from each other within the semantic space.

In Figure 1 the word embeddings of two short texts are represented as dots in a two-dimensional space. As can be observed from the picture, the two texts have terms that are close to each other (at the top and bottom in the figure), while the ones at the far left and right have no counterpart in their immediate vicinity. Regardless of this discrepancy, the means of word embeddings of the two phrases are close to one another. The fact that both texts have a term unlike any term in the other text is not well represented by the means. So calculating mean or some sort of measure like that for a phrase is not going to help much in calculating semantic similarity between word phrases.

  
Figure 1: Hypothetical example — two-dimensional representation of the word embeddings for two short texts

In this figure two-dimensional representation of word embeddings for two short texts are depicted, both consisting of three words and represented as transparent and opaque dots respectively. And their mean is represented as transparent and opaque crossed dots respectively. As it clearly shows mean position of word embedding as a representation for phrases can be misleading.

To transform from word-level similarity to text-level similarity, a lot of studies has been carried out in the past few years such as [10], [12], [16], [17], [18] which span over various strategies from embedding phrases and sentences to applying an additional layer to the system.

In [10] it proposes a system that uses IDF( Inverse Document Frequency) as an indication of the contribution of each word to the meaning of short text phrases And uses a supervised learning model based on support vector machines to train a model on a sentence pair modelling task and archive a 76.6% accuracy over Microsoft Research Paraphrase Corpus.

In the paper [19] the researchers have taken a novel approach and used a novel approach and used an LSTM(Long Short Term Memory) encoder to encode short text snippets and measure the semantic similarities completely through an unsupervised learning approach. Thus this approach can not directly cooperate with the study as it does not need the requirement of classifying the text into a relevant keyword.

In [20] introduce a new model called Embedding- based Topic Model (ETM), for identifying the key topics underlying a collection of short texts. They built the distributed word embeddings for the vocabulary of the collection. After, they aggregated the short texts into long pseudo-texts by incorporating the semantic knowledge from word embeddings. They implemented K-means using the Word Mover’s Distance, to compute the distance between two short texts.

Deep Structured Semantic Models (DSSM) [11] is a well-known short text matching technique. DSSM represents text as vectors, and the vector distance between them is the matching score they get. DSSM uses three kinds of neural networks to represent text:

* Feed Forward Network-based DSSM
* Convolutional Network-based DSSM
* LSTM-DSSM\*

## 2.5 Problem with the traditional word embeddings and the solution

However, traditional word embeddings from algorithms like word2vec or glove have two major drawbacks, First one is not being a full language model. Those word embeddings are just a mathematical representation of word semantics. So it is just going to output some vector for individual words and it has no way of comparing the semantics of two phrases. The other main drawback is word representation of these word embeddings are not context-sensitive. For example consider these two phrases, “my Apple iPhone” and “eat an Apple” in these two instances word apple clearly has different semantics, at the first time representing the corporation apple and in the second instance it represents the fruit apple, despite having two vector representation for those two instances word embeddings gives the same vector for both occasions.

Bidirectional Encoder Representation from Transformers(BERT) [21] is using the bidirectional context of words to capture the meaning of words. Pre-trained BERT models can be combined with one additional layer to compute the semantic similarity between texts. As BERT is using transformers for encoding text instead of LSTM encoders the process of encoding text can be done in parallel for each word, instead of sequentially, encoding words like in LSTM. And BERT as being a language model it can consider the bidirectional context of the input text. And it can also create a context-sensitive representation for words considering the bidirectional context of a word.

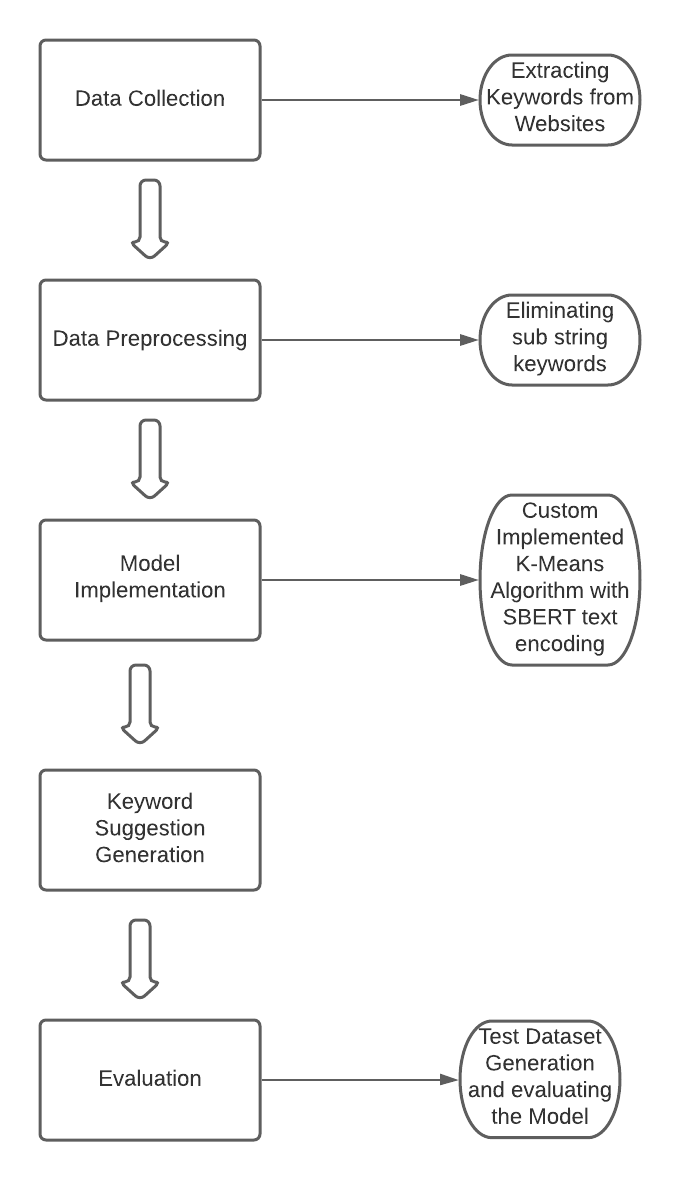
As the BERT model lacks the two main drawbacks of the traditional word embeddings, So BERT is a much better candidate for NLP problems than traditional word embeddings. In fact, BERT has been able to archive the state of the art performance in many NLP tasks. In [22] researchers claim that BERT is significantly undertrained and they purpose a model with an improved recipe for training which is called RoBERTa which further improves the performance of the BERT model.

But either BERT or RoBERTa is not ideal for semantic matching, it requires both sentences to be fed into the model at once and it create a massive computational overhead [23]. In [23] purpose Sentence-BERT(SBERT) which is a modification of pre-trained BERT network that use siamese triplet network structure to derive semantically meaningful sentence embeddings that can be compared using a distance measure such as cosine distance.

# Design

This chapter describes the overall proposed design of the research. It consists of three main sections as Data Collection and Preprocessing, Model implementation and Keyword suggestion generation and finally the Evaluation. Data Collection and Preprocessing describe how does the data collection happens and what are the preprocessing steps taken place. The Model implementation and Keyword suggestion generation describe the architecture of the created model and how it generates keyword suggestions for a given keyword phrase and finally in the evaluation section it describes how does the above-created model get evaluated.

The following figure shows the high-level architecture of the research design.

  
Figure 2: High-level architecture of the research design

## 3.1 Data Collection and Preprocessing

The Internet is a huge space, So expanding this study over every niche of websites is not practical with this study. So websites of the niche “Android” have been chosen to carry out the research because it meets the following challenges of the study can be met with the “Android” niche.

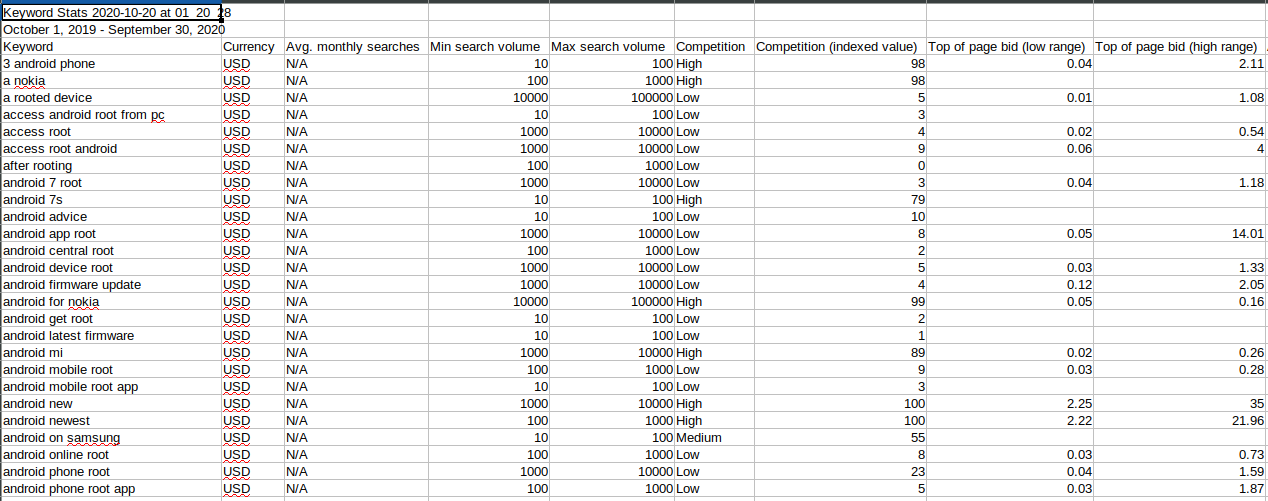
* Need a domain understanding in creating datasets
* Has a large amount of content and a large audience that consumes the content
* Vocabulary used in the field changes rapidly

So the research begins with the data collection. In the process of data collection, a dataset consisting of keywords related to the Android niche must be created. The easiest and the most effective way of creating that dataset is extracting keywords from the top websites that publish content under the Android niche.

The following websites were selected for data collection through the sources [24] and [25]. And they were verified to be the best fits to the task by using the Alexa traffic ranking [26] of each website.

1. Android Police - www.androidpolice.com
2. Android Headlines - https://www.androidheadlines.com/
3. Android Authority - https://www.androidauthority.com/
4. Android Guys - https://www.androidguys.com/
5. Android Advices - http://androidadvices.com/
6. Droid Life - https://www.droid-life.com/
7. Android Central - www.androidcentral.com
8. MakeUseOf - https://www.makeuseof.com/
9. Bestforandroid - https://bestforandroid.com/
10. Android Authority
11. Android Central
12. Mobile internist - https://mobileinternist.com/
13. XDA - https://www.xda-developers.com/
14. XDA forum - <https://forum.xda-developers.com/>

A dataset of keywords about the Android niche is created using keywords extracted from top websites about Android using Adword Keyword Planner [2]. By combining all the unique keywords from all the websites a dataset of keywords about android was created which consisted of 7167 keywords.

  
Figure 3: Screenshot of exported keywords from Google Keyword Planner

Search keywords are word phrases that repeatedly appear within multiple search queries. So within the created dataset, there are some keywords that are useless and not provide any value to the SEO process. As an example within these two keywords “new android phone” and “cheap android phone” on both occasions the term “android phone” appears. So the term “android phone” can also be identified as a keyword that has a higher frequency than both of the keywords it is descended from.

So eliminating those keywords in advance is useful as those keywords are useless substrings of bigger keywords. So in the data preprocessing we eliminate all the keywords that can be found as a substring of a bigger keyword. So after this preprocessing step, the dataset shrinks to a dataset with 4528 keywords which is the final dataset that the model is going to be trained on.

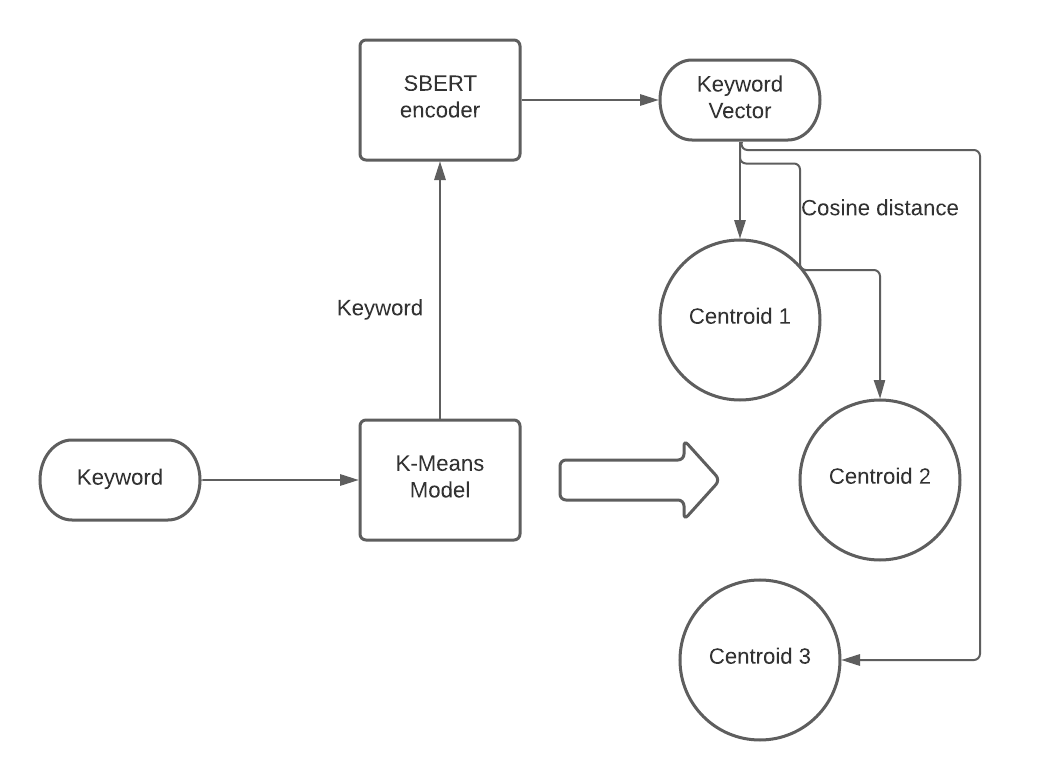
## 3.2 Model Implementation and Keyword Suggestion generation

The proposed solution consists of two parts. Word embeddings are obtained by unsupervised learning and a classifier to classify word phrases to the keyword it belongs to. Both keywords and phrases to be matched needed to be encoded to a mathematical representation of the text phrases using the most suited mechanism described in the literature, so the pre-trained SBERT model “stsb-roberta-base” which is based on the RoBERTa network that has been fine-tuned for semantic matching is used to encode the text phrases.

By obtaining a vector representation of keywords via the SBERT model it is technically feasible to compare the semantic distance between the keywords and any candidate text phrase via a distance measure such as euclidean distance. So there are two basic approaches to generate a keyword suggestion that is semantically similar to a particular text phrase. First, one encodes the text phrase into a vector representation via the SBERT model and encode each keyword into there respective vector representation via the SBERT model and compare the distance between the text phrase and each keyword and selecting the keyword with the least distance as the suitable keyword. And the second approach is encoding all the keywords in advance and holding their vectors in memory to compare those vectors with the vector of the candidate text phrase.

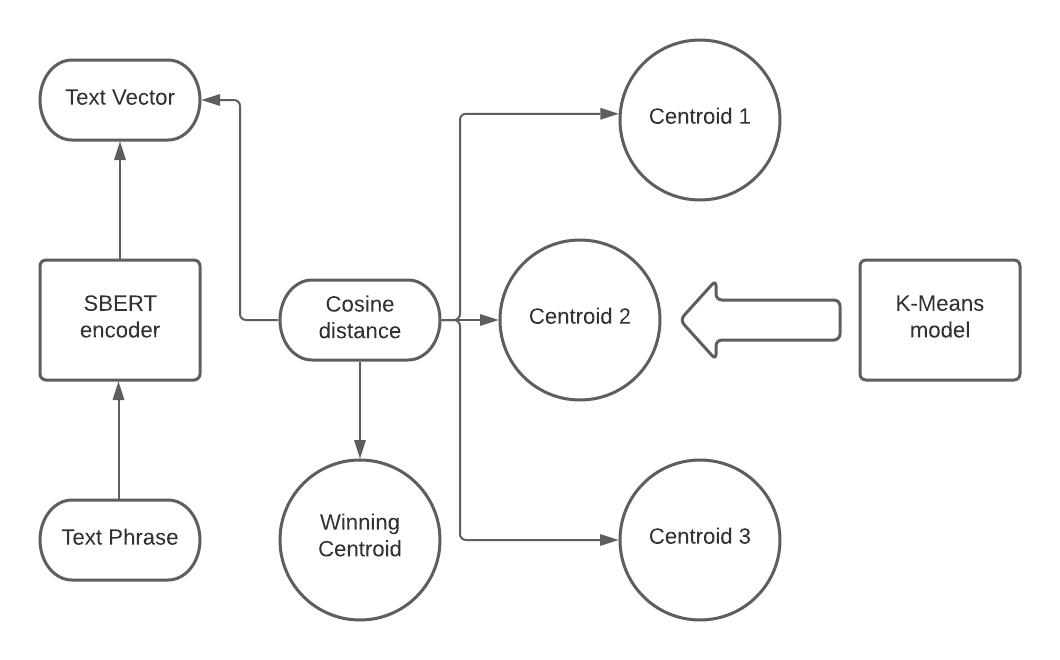
Both these methods come with their own issues, in the first methods we have to encode the keywords over and over again for each candidate text phrase, and also it is a very time-consuming process as the number of keywords in huge, so it takes a long time to compute the best keyword for a candidate text phrase. In the second approach as the number of keywords is large it is difficult to contain all the vectors of keywords in the memory and also it is a very time consuming to compare the vector of text phrase with each keyword vector via a distance measure such as euclidean distance or cosine distance with each keyword and compute the keyword with the least distance. So both of these approaches are not suitable for real-world use.

So this research proposes a novel approach with a custom implementation of a K-Means algorithm. So that process is as follows, first select the suitable number of centroids for the K-Means model and then initialize each centroid with a vector of a randomly selected keyword. And cluster the keywords into the number of selected centroids. And then train the model until it converges. Here within the clusters, the K-Means model holds the keywords as raw keywords, not their vector representations, hence it does not overload the memory and it only needs the vectors of centroids to be held in the memory. Thus in the training process, it encodes the keywords into vector representation when it is needed to be clustered and does not hold on to the memory.

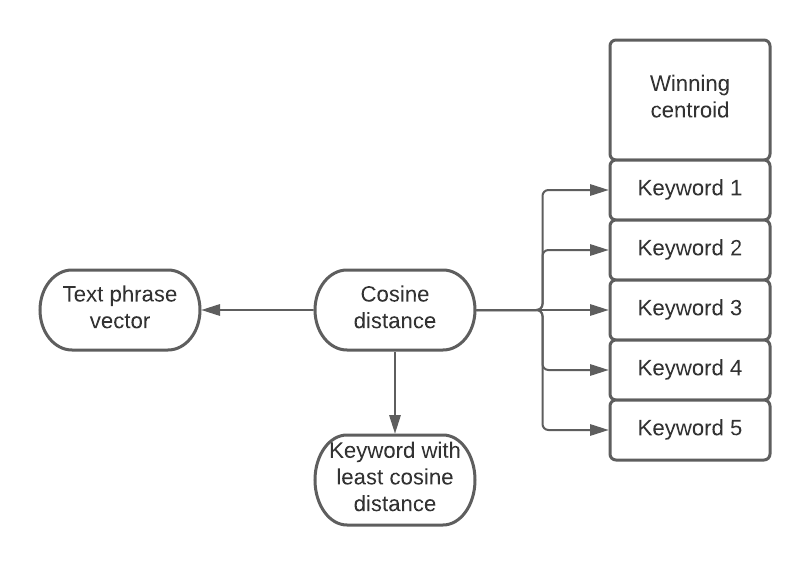
  
Figure 4: Training Process of K-Means model

This figure tries to depict the training process of the K-Means model. When a keyword is presented to the model it sends it to the SBERT model to create a vector representation of the keyword, And then that vector is compared with the vectors of each centroid by calculating the cosine distance between them and select the centroid it belongs to as the centroid shares the least distance with a vector of the keyword. After assigning all the keywords to centroids, the vector of centroids gets replaced by the average of the vectors of the keywords that are assigned to each centroid. And this process continues until the model converges.

So now this model can be used to generate keyword suggestions for a candidate text phrase with a very little amount of computations and less time. So in the generation of keyword suggestions from this model. First, the candidate text phrase gets encoded into its vector representation using the SBERT model. And the cosine distance for each centroid of the K-Means model gets calculated and selects a winning centroid with the least centroid distance.

  
Figure 5: Selecting the Winning Centroid

Then the text phrase gets compared against each keyword within the winning centroid. The cosine distance between the text phrase and each keyword gets calculated. The keyword with the lowest distance with the text phrase is selected as the optimal keyword suggestion for that text phrase.

  
Figure 6: Selecting the best matching keyword from winning centroid

## 3.3 Evaluation

The evaluation of the proposed solution is proposed to be carried on using a test dataset. For that, a labelled dataset of text phrases with their corresponding keyword was created. So the accuracy of the created model was calculated using the labelled dataset created. Further details on the evaluation process will be discussed in chapter 5.

# Implementation

This chapter is going to elaborate on the implementation details of the research design. This chapter can be examined according to two main sections as Dataset creation and preprocessing and Model Implementation.

## 4.1 Dataset creation and preprocessing

First, the keyword sets were extracted using the Google Adword Keyword planner as mentioned earlier. Then for the dataset creation and the preprocessing of the dataset python script was used. In that python script, the pandas library was used to edit and create the CSV files.

So in the dataset creation step, all the unique keywords from all keyword files were extracted and appended to one file to create the keyword dataset of the study using the python script.

So as the preprocessing step all the keywords which can be identified as a substring of another keyword was identified and eliminated by exhaustively searching through the dataset using a python script.

## 4.2 Model Implementation

The implemented model is consist of two main parts, a text phrase encoder that generates a vector representation for a given text phrase and a custom implementation of the K-Means algorithm that is responsible for efficiently finding out the best semantically matching keyword for a given text phrase using the text phrase encoder.

The text phrase encoder was implemented using the sentence\_transformer library which is available on Github as a result of the SBERT implementation. sentence\_tranformer library use pre-trained BERT models to create the semantic vector representation of text phrases. It has a function called encode which outputs the vector representation of imputed text phrase, which is encoded using the selected pre-trained BERT model.

The pre-trained model “stsb-roberta-base” was selected as the underlying BERT model of the encoder which is based on the RoBERTa algorithm and which has also been fine-tuned for semantic matching tasks. The pre-trained model “stsb-roberta-large” also is a good candidate for the task, which is also based on the RoBERTa algorithm, and it might make even better results than the “stsb-roberta-base” model where it creates a vector of 1024 dimensions instead of the 768 dimension vector created by the selected “stsb-roberta-base” model. But due to limitations of the local machine environment “stsb-roberta-base” model was selected as the model for text semantic encoding.

As mentioned in the research design a custom implementation of the K-Means algorithm is used to cluster the keyword set into clusters to reduce the time and memory complexity of the problem of finding the optimal keyword suggestion for a given text phrase.

In that K-Means implementation, a separate class was created for keywords called KeywordItem which is responsible for supplying the keyword and the encoded vector for keywords when requested by the K-Means model. There only one member was available that was for keyword and every time the K-Means model request for the semantic vector of the keyword, KeywordItem object call the text phrase encoder and keyword text phrase and send that vector for the K-Means model. That way the K-Means model was able to cluster the keyword set with less memory use without causing a memory overflow.

So using the above-mentioned tactic the K-Means model was able to make this solution feasible by reducing the memory complexity of the problem. However, still, the K-Means model was needed to hold the vectors of centroids in the memory. Which makes the number of centroids that can be used in the local machine to be around 200 centroids. However, the ideal amount of centroid can vary from the value we choose. The ideal value for centroids can be identified by the following calculations.

When

C - is the number of centroids

K - is the number of total keywords

te - is the time for encoding a text phrase as a vector

td - is the time for calculating the cosine distance between two vectors

T - the total time spent on calculating a keyword suggestion for a text phrase

The value T can be approximated as a function of C as follows

  
Figure 7: Equation for calculating the total time for suggesting a keyword for a text phrase

So by minimizing the value of T we can gain the ideal number of C(number of centroids) which turns into the following equation.

  
Figure 8: Equation for number of centroids

Where the experiment values of the te, td, with the value of K shows a value near 1300 as the ideal number of centroids.

So if there was a much more powerful machine environment to run the model, most time-efficient results could be obtained by using a centroid count of around 1300, but due to the limitations of the local machine environment, the number of centroids was limited to 200.

When comparing the distance between semantic vectors the cosine distance was calculated between the two vectors. There is no clear theoretical reason for choosing cosine distance over euclidian distance or any other distance measure. But most of the semantic matching tasks hat described in the literature which is based on distributional semantics tend to use cosine distance over other distance measures, and experiments also show better results with cosine distance over other distance measures.

Another important detail about the implementation of the K-Means model is the “SEMANTIC\_DISTANCE\_THRESHOLD”. This is a constant value that was implemented using the best value from the experimental results. This value stands as a threshold to be the maximum value that can be taken by the cosine distance of two vectors that can be considered to be semantically similar. So as an example when we calculate the best keyword suggestion for a particular text phrase, if the distance between the semantic vector of the text phrase and the best matching keyword is bigger than the “SEMANTIC\_DISTANCE\_THRESHOLD” then that keyword gets discarded and not get considered as a semantically similar keyword to the text phrase. This construct of thresholding the distance between two vectors stop the model from predicting keywords totally unrelated text phrases. And it is also useful when suggesting multiple keywords that can be semantically similar to a text phrase.

# Results and Evaluation

The intrinsic evaluation is planned to be carried out for the model. And the model can be evaluated with the results of the testing data. As it had had already mentioned earlier a labelled dataset of text phrases with their corresponding keywords has been created for the evaluation process of the model.

So as the evaluation of the model, accuracy over the test dataset can be calculated. The test dataset is consist of 987 text phrases typed against keywords that are semantically similar. So for any text phrase, the best keyword suggests by the implemented model and it’s labelled keyword can be compared to be the same or not. By that comparison, the accuracy of the model can be calculated using the total number of text phrases in the dataset and the number of text phrases that are correctly guessed by the implemented model.

But still, there is a problem if the evaluation carried out like that, The keywords are not semantically distinct. So there are overlapping between the semantics of the keywords. So as an example, “1 ui samsung” and “one ui for samsung” both presents in the keyword dataset, as both of these keyword shares almost similar semantics, so in case of a text phrase that describes one of these keywords, the model can choose either of these keywords despite their label. On such occasions, the prediction of the model can be identified as false as it does not obtain the labelled output though it has correctly identified the semantic relationship between the text phrase and the keyword.

So to overcome this issue another step in the evaluation is proposed, Which includes the model suggesting multiple keywords on behalf of a text phrase. Here the model keeps a semantic threshold value based on the experimental results when comparing the cosine distance between the text phrase vector and the vectors of the keywords, all the keywords that sit below the threshold value get selected as keyword suggestions for the text phrase.

So when calculating the accuracy with this new multiple keyword suggestion approaches, the model generates the multiple keyword suggestions for an input text phrase, if any of the guessed keywords is the labelled keyword that text phrase gets considered as a correct guess from the model.

These two measures, the accuracy of the model when predicting single keywords and the accuracy of the model when predicting multiple keywords, can be identified as the lower bound and the upper bound of the predictive power of the model.

The experimental results of the model according to those two evaluation criteria are as follows.

Table 1: Results of evaluating the model

|  |  |
| --- | --- |
| Single Keyword suggestion accuracy | Multiple Keyword suggestion accuracy |
| 62.26% | 74.46% |

# Conclusions

## 6.1 Introduction

This thesis is written on developing an NLP model on Identifying text phrases containing similar semantic meaning with search keywords. Initially, the research started with an in-depth review of the literature of SEO and semantic matching techniques available in the literature. Chapter 2 describes the literature review conducted on behalf of understanding the existing knowledge in the domain and understand the gaps in this research area.

As a result of the literature review carried out it was evident that no research has been publically carried out on the identifying text phrases containing similar semantic meaning with search keywords.

This chapter describes an overall picture of the conclusion drawn from this study. It can be further divided into four sections, Conclusions about research questions, Conclusions about the research problem, Limitations and Implications for further research.

## 6.2 Conclusions about research questions (aims/objectives)

The main objective of the study was to explore the possibility of using existing NLP techniques to find alternative keyword phrases for already existing word phrases within the content of webpages. So from this study, it was clear that already existing NLP technologies can be used to archive this task.

The research questions are as follows, the first one was, How to identify word phrases that contain a similar meaning with keywords related to content? Here the literature review shows that semantic matching techniques that are described in the NLP are the way to capture the similarity between keywords and word phrases.

Then the next question was, What are the techniques available to compare the similarity of texts? Here we explore the existing techniques for the semantic matching and was able to identify the pro and cons of each semantic matching techniques and most suitable domains for each of those techniques can be used.

So the next question was, Which of those are applicable to compare semantic similarity between small word phrases? This shows that most of the semantic matching techniques except techniques based on distributional semantics are not suitable for semantic matching for short text phrases. Especially the semantic matchings techniques that are based on lexical matching need the matching texts to be longer than short text phrases.

Finally, the question that arises was, How to develop a model that can identify similarities between short text phrases and search keywords from those applicable techniques? As we discussed deeply in chapter 3 about the implementation of the model, This study was able to construct a method to identity search keywords that share similar semantic meaning with search keywords using the SBERT model with the combination of K-Means clustering. With a lot of experiments and fine-tuning, the results mentioned in chapter 5 was able to be archived. The results archived by the study can be considered as good results as this is study is the first study of this particular domain.

## 6.3 Conclusions about research problem

As mentioned in chapter 1, this research is about applying NLP techniques to content SEO. However, there is very little research that has been carried on using NLP to improve web content SEO. Thus, this study, identifying text phrases containing similar semantic meaning with search keywords using NLP, is a great contribution to the research area of improving web content SEO using NLP. And it is believed that this research will inspire more researchers to conduct research on this particular domain. In addition to addressing the research problem, contributions to this research also resulted in creating two datasets of android keywords and labelled text phrases with android keywords which might be beneficial in future studies or anyone interested in the subject as further studies and experiments can be easily carried on these datasets easily.

## 6.4 Limitations

The scope of the study had been shrunken to match the timelines of the study. So bigger dataset for training and also evaluation of the model might make a much better picture of the evaluated model. Since as it is already mentioned dataset was needed to be created from scratch by extracting keywords from websites and manually typing text phrases. Creating large datasets that entirely cover any category of website is quite hard with the given time frame. The results of the study could be more robust if much bigger datasets were used.

The model training and all the executions were done on the local machine. Even though the hardware of the local machine was minimally sufficient for the requirements there were limitations exposed by the local machine environment. As an example, the ideal number of centroids for the K-Means model was higher than the number of centroids used in the study. But the memory limitations of the local machine does not allow to increase the number of centroids to be the ideal number. And the long time taken for the training process of the model was also a major limitation from the local machine environment.

## 6.5 Implications for further research

Many investigations can be carried out to develop and train more robust and improved models with the help of high-end hardware resources by changing the text phrase encoding model such as using different trained BERT models for the encoding and also changing the fine-tuning parameters of the BERT models. And It can also possible to train a BERT model in corpus created especially for this task. Further experiments can be carried out on the K-Means clustering part such as changing the number of centroids etc. It can also experiment with a completely novel approach except the method based on the K-Means algorithm which is used in this study. Experiments can be done on much bigger datasets for more robust interpretations of results.

This study glimpse on a research area that is not much popular in NLP as well, the semantic matching between short text phrases haven’t archived much of the attention of the researchers. So the method described in this study or variation of this method can be also used to other problems that semantic similarity of short text phrases can be applied to.

# References

[1] T. Mavridis and A. L. Symeonidis, “Identifying valid search engine ranking factors in a Web 2.0 and Web 3.0 context for building efficient SEO mechanisms,” *Eng. Appl. Artif. Intell.*, vol. 41, pp. 75–91, May 2015, doi: 10.1016/j.engappai.2015.02.002.

[2] “Keyword Planner - 272-369-4930 - Google Ads.” https://ads.google.com/aw/keywordplanner/home?ocid=110535901&euid=115074181&\_\_u=5045184669&uscid=110535901&\_\_c=2656784949&authuser=0&subid=lk-en-ha-aw-bk-c-l00%21o3~Cj0KCQjw6PD3BRDPARIsAN8pHuGOdL8MdP1bVt5K\_ZPA37DkJsBF9KJPvBlGBhtRZyjXKe8sTGmOHLgaAuFAEALw\_wcB~76455040501~kwd-60278213695~6806576082~389349835136 (accessed Jul. 02, 2020).

[3] “Search Engine Optimization (SEO) Starter Guide - Search Console Help.” https://support.google.com/webmasters/answer/7451184?hl=en (accessed May 16, 2020).

[4] “Webmaster Guidelines - Bing Webmaster Tools.” https://www.bing.com/webmaster/help/webmaster-guidelines-30fba23a (accessed May 16, 2020).

[5] A. Dixit and M. Agarwal, “Hybrid Approach to Search Engine Optimization (SEO) Techniques,” vol. 1, no. 3, p. 5, 2015.

[6] Y. S. Business, “Your SEO Checklist: 4 Steps to Optimizing Your Website,” *Business Resource Center*, Feb. 06, 2020. https://smallbusiness.yahoo.com/advisor/resource-center/seo-checklist-4-steps-optimizing-website-170000741/ (accessed May 16, 2020).

[7] H. Banaei and A. R. Honarvar, “Web page rank estimation in search engine based on SEO parameters using machine learning techniques,” p. 6, 2017.

[8] Rakesh Kumar and S. Saini, “A Study on SEO Monitoring System Based on Corporate Website Development,” *Int. J. Comput. Sci. Eng. Inf. Technol.*, vol. 1, no. 2, pp. 42–49, Jun. 2011, doi: 10.5121/ijcseit.2011.1204.

[9] “SEO Content Analysis API - SEO Score API - SEO Review Tools.” https://www.seoreviewtools.com/seo-content-analysis-api/ (accessed May 17, 2020).

[10] T. Kenter and M. de Rijke, “Short Text Similarity with Word Embeddings,” in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management - CIKM ’15*, Melbourne, Australia, 2015, pp. 1411–1420, doi: 10.1145/2806416.2806475.

[11] P. Aghahoseini, “Short Text Similarity: A Survey,” p. 6.

[12] Q. Le and T. Mikolov, “Distributed Representations of Sentences and Documents,” p. 9.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space,” *ArXiv13013781 Cs*, Sep. 2013, Accessed: Jul. 01, 2020. [Online]. Available: http://arxiv.org/abs/1301.3781.

[14] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, “Indexing by latent semantic analysis,” *J. Am. Soc. Inf. Sci.*, vol. 41, no. 6, pp. 391–407, 1990, doi: 10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9.

[15] J. Pennington, R. Socher, and C. Manning, “Glove: Global Vectors for Word Representation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, 2014, pp. 1532–1543, doi: 10.3115/v1/D14-1162.

[16] C. Banea, D. Chen, R. Mihalcea, C. Cardie, and J. Wiebe, *SimCompass: Using Deep Learning Word Embeddings to Assess Cross-level Similarity*. .

[17] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality,” in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 3111–3119.

[18] Y. Kim, “Convolutional Neural Networks for Sentence Classification,” *ArXiv14085882 Cs*, Sep. 2014, Accessed: Jul. 02, 2020. [Online]. Available: http://arxiv.org/abs/1408.5882.

[19] L. Yao, Z. Pan, and H. Ning, “Unlabeled Short Text Similarity With LSTM Encoder,” *IEEE Access*, vol. 7, pp. 3430–3437, 2019, doi: 10.1109/ACCESS.2018.2885698.

[20] J. Qiang, P. Chen, T. Wang, and X. Wu, “Topic Modeling over Short Texts by Incorporating Word Embeddings,” *ArXiv160908496 Cs*, Sep. 2016, Accessed: Oct. 30, 2020. [Online]. Available: http://arxiv.org/abs/1609.08496.

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *ArXiv181004805 Cs*, May 2019, Accessed: Nov. 05, 2020. [Online]. Available: http://arxiv.org/abs/1810.04805.

[22] Y. Liu *et al.*, “RoBERTa: A Robustly Optimized BERT Pretraining Approach,” *ArXiv190711692 Cs*, Jul. 2019, Accessed: Jan. 12, 2021. [Online]. Available: http://arxiv.org/abs/1907.11692.

[23] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” *ArXiv190810084 Cs*, Aug. 2019, Accessed: Jan. 12, 2021. [Online]. Available: http://arxiv.org/abs/1908.10084.

[24] A. Anil, “Android blogs: Top 10 websites for tips on android, news, best apps and reviews (2020 list),” *Medium*, Jun. 22, 2020. https://medium.com/@arjunanilrustaq/android-blogs-top-10-websites-for-tips-on-android-news-best-apps-and-reviews-2020-list-4e42af9039c5 (accessed Oct. 29, 2020).

[25] “Best Android News Sites 2020 - Popular Android Blogs & Twitter Follows,” *alltop.com*. http://www.alltop.com (accessed Oct. 29, 2020).

[26] K. Cooper, “Keyword Research, Competitor Analysis, & Website Ranking | Alexa,” *Alexa.com*. https://www.alexa.com/ (accessed Oct. 29, 2020).

# Appendix A: Code Listings

## A.1 Code snippet of Dataset creation and Preprocessing

import pandas as pd

import numpy as np

def fimport(filename,sep=',',encoding='utf-16',skiprows=1, header=0):

frame = pd.DataFrame()

frame = pd.read\_csv(filename,sep=sep, skiprows=skiprows, header=header, encoding=encoding, error\_bad\_lines=False)

return frame

filenames = ['Android Advices Keyword Stats 2020-10-20 at 01\_20\_28.csv'

, 'Android Authority Keyword Stats 2020-10-20 at 01\_26\_08.csv'

, 'Android Central Keyword Stats 2020-10-20 at 01\_15\_24.csv'

, 'Android Guys Keyword Stats 2020-10-20 at 01\_24\_13.csv'

, 'Android Headlines Keyword Stats 2020-10-20 at 01\_27\_46.csv'

, 'Android Police Keyword Stats 2020-10-20 at 01\_29\_25.csv'

, 'Appedit mobile Keyword Stats 2020-10-18 at 04\_24\_41.csv'

, 'Best for Android Keyword Stats 2020-10-18 at 04\_36\_16.csv'

, 'Droid Life Keyword Stats 2020-10-20 at 01\_17\_48.csv'

, 'Make Use of Keyword Stats 2020-10-20 at 00\_44\_17.csv'

, 'Mobile internist Keyword Stats 2020-10-18 at 04\_34\_29.csv'

, 'XDA Forum Keyword Stats 2020-10-18 at 04\_28\_55.csv'

, 'XDA Keyword Stats 2020-10-18 at 04\_26\_36.csv']

uniqueKeywords = []

for fileName in filenames:

df = fimport(fileName)

length = len(df)

for i in range(0, length):

isAlreadyAdded = False

keyword = df.loc[i,'Keyword']

for j in range(0, len(uniqueKeywords)):

if uniqueKeywords[j] == keyword:

isAlreadyAdded = True

if (isAlreadyAdded):

continue

else:

uniqueKeywords.append(keyword)

uniqueKeywords.sort()

subSetRemovedKeywords = []

def isSubphrase(subphrase,phrase):

subphraseArray = subphrase.split(" ")

phraseArray = phrase.split(" ")

for word in subphraseArray:

if word in phraseArray:

continue

else:

return False

return True

def removeSubstringKeywords(keywords, newKeywords):

for i in range( 0 , len(keywords)):

keyword = keywords[i]

if (len(newKeywords) == 0):

newKeywords.append(keyword)

continue

canAppendKeyword = True

newKeywordsLength = len(newKeywords)

for j in range ( 0, newKeywordsLength):

newKeyword = newKeywords[j]

if isSubphrase(keyword,newKeyword):

canAppendKeyword = False

break

if isSubphrase(newKeyword,keyword):

newKeywords.remove(newKeyword)

newKeywordsLength -= 1

break

if ( canAppendKeyword ):

newKeywords.append(keyword)

removeSubstringKeywords(uniqueKeywords,subSetRemovedKeywords)

import csv

file = open('unique.csv','w', newline='')

with file:

header = ['Keyword','Text Phrase']

writer = csv.DictWriter(file, fieldnames = header)

writer.writeheader()

# writing data row wise to the csv file

for keyword in uniqueKeywords:

writer.writerow({header[0] : keyword,

header[1] : 'Not Set'})

file.close()

file = open('uniqueSubSetRemoved.csv','w', newline='')

with file:

header = ['Keyword','Text Phrase']

writer = csv.DictWriter(file, fieldnames = header)

writer.writeheader()

# writing data row wise to the csv file

for keyword in subSetRemovedKeywords:

writer.writerow({header[0] : keyword,

header[1] : 'Not Set'})

file.close()

file = open('keywordset.csv','w', newline='')

with file:

header = ['Keyword']

writer = csv.DictWriter(file, fieldnames = header)

writer.writeheader()

# writing data row wise to the csv file

for keyword in subSetRemovedKeywords:

writer.writerow({header[0] : keyword})

file.close()

## A.2 Code snippet of semantic encoding of text phrases

from sentence\_transformers import SentenceTransformer

# importing the bert model for sentence encording

model = SentenceTransformer('stsb-roberta-base')

# function to encode sentences

def encodeSentence(sentence):

sentences = [sentence]

sentence\_embedding = model.encode(sentences)

return sentence\_embedding

## A.3 Code snippet of custom K-Means Implementation

from random import random

import numpy as np

from scipy import spatial

import json

from sbert\_sentence\_encording import encodeSentence

# String to return when no keyword is matching

NO\_KEYWORD\_MATCHING\_STRING = "No Keyword Matching"

SEMANTIC\_DISTANCE\_THRESHOLD = 15.0

# This function is used to load a previouly saved model as a json file

def loadModelFromFile(filename):

# opening the target json file

with open(filename) as json\_file:

model = json.load(json\_file)

# fetching the information about kmeans model

dimetionality = model['dimetionality']

noOfCentroids = model['noOfCentroids']

centroidsObject = model['centroids']

# Creating a list for centroids and iterating over centroid item in json file

centroids = []

for centroidItem in centroidsObject:

# Fetch the details of centroid

id = centroidItem['id']

dimetionality = centroidItem['dimetionality']

cordinatesArray = centroidItem['cordinates']

itemsArray = centroidItem['items']

# creating a cordinate vector inizaially with zeros

cordinates = np.zeros(shape=(1,dimetionality))

# Assigning cordinate values iterating through the cordinates

for i in range(0,len(cordinatesArray)):

cordinates[0][i] = np.float32(cordinatesArray[i])

items = []

# Assigning items values iterating through the items

for i in range(0,len(itemsArray)):

item = KeywordItem(itemsArray[i])

items.append(item)

# Creating centroid object from fetched data and appending to centroid list

centroid = Centroid(dimetionality, id, cordinates)

centroid.setItems(items)

centroids.append(centroid)

# Creating and returning the kmeans model from centroid list and fetched data

kmeansModel = KMeans(noOfCentroids,dimetionality)

kmeansModel.setCentroids(centroids)

return kmeansModel

# This function is for identifing the index of minimum value of a python list

# This approch saves time by only iterating once at array, instead of arr.index(min(arr))

def indexOfMinimum(listObj):

# If list if empty return None

if len(listObj) == 0:

return None

# Else iterate and find the min and index of min and return index

else:

min = listObj[0]

index = 0

for i in range(1,len(listObj)):

if (listObj[i] < min):

min = listObj[i]

index = i

return index

# Kmeans class

class KMeans:

# Constructor of the kmeans class

def \_\_init\_\_(self,noOfCentroids, dimentionality, centroidCordinates = None):

self.dimetionality = dimentionality

self.noOfCentroids = noOfCentroids

self.centroids = []

# if centroid cordinates are not given create centroids with random inizial values

if (centroidCordinates == None):

for i in range(0,self.noOfCentroids):

centroid = Centroid(dimentionality, i)

self.centroids.append(centroid)

# if centroid cordinates are given create centroids with given cordinates

else:

for i in range(0,self.noOfCentroids):

centroid = Centroid(dimentionality, i,centroidCordinates[i])

self.centroids.append(centroid)

# This function is used when loading the model from a file to manually set centroids

def setCentroids(self,centroids):

self.centroids = centroids

# Adding items to the correct centroid with the least distance

def addItemToCentroid(self,item):

distances = []

itemVector = item.getVector()

# Calculating the distance with each centroid

for centroid in self.centroids:

centroidVector = centroid.getDimenVector()

distance = spatial.distance.cosine(itemVector, centroidVector)

distances.append(distance)

# Getting the id of neighrest centroid and adding the item to that centroid

indexOfMin = indexOfMinimum(distances)

self.centroids[indexOfMin].addItem(item)

# method to train the dataset on the kmeans model

def fit(self,items):

isCentroidsAdjustementFinished = True

# Run the loop until clusters converge and break the loop

while True:

# At each iteration inizialy set the adjustment finish to true

isCentroidsAdjustementFinished = True

# Add all the items to respective centrooids

for item in items:

self.addItemToCentroid(item)

# After adding the items adjust the centroids to mean of items

for centroid in self.centroids:

temp = centroid.adjustCentroidCordinates()

if (not temp):

# if any of the centroid within the iteration is not in the mean position set adjustmentFinished to false

isCentroidsAdjustementFinished = False

if ( isCentroidsAdjustementFinished):

break

# Setting the items in the centroids to empty array again at the end of iteration

for centroid in self.centroids:

centroid.resetItems()

# Experimental method to print the architecture of the kmeans model to verify

def printModel(self):

print("\*\*\*\*\* Printing the Model \*\*\*\*\*")

print("No of centroids " , self.noOfCentroids)

# Iterating over all centroids and printing centroid details

for centroid in self.centroids:

print("Centroid ", centroid.id)

print("Elements in centroid")

for item in centroid.items:

print(item.getLabel())

print("")

# method to save the trained model as a json file

def saveModelToAFile(self,filename):

# create a dictionary object and adding model details to it

model = {}

model['dimetionality'] = self.dimetionality

model['noOfCentroids'] = self.noOfCentroids

model['centroids'] = []

# make a list for centroids and adding each centroid as a new dic object to the list

for centroidItem in self.centroids:

centroid = {}

centroid['id'] = centroidItem.id

centroid['dimetionality'] = centroidItem.dimetionality

centroid['cordinates'] = []

for value in centroidItem.cordinates[0]:

centroid['cordinates'].append(str(value))

centroid['items'] = []

for item in centroidItem.items:

centroid['items'].append(item.getLabel())

model['centroids'].append(centroid)

# Dumping the dic as a json file to the file

with open(filename,'w') as output:

json.dump(model,output)

# Method to getting the maching keyword from the model

def getMatchingKeyword(self,phrase):

encording = encodeSentence(phrase)

# getting the winiing centroid

centroidDistances = []

for centroid in self.centroids:

centroidVector = centroid.getDimenVector()

distance = spatial.distance.cosine(encording, centroidVector)

centroidDistances.append(distance)

winnerCentroidId = indexOfMinimum(centroidDistances)

winnerCentroid = self.centroids[winnerCentroidId]

# getting the most simillar keyword item from the centroid

keywordDistances = []

for keywordItem in winnerCentroid.items:

itemVector = keywordItem.getVector()

distance = spatial.distance.cosine(encording, itemVector)

keywordDistances.append(distance)

winnerKeywordId = indexOfMinimum(keywordDistances)

winnerKeyword = winnerCentroid.items[winnerKeywordId]

# Semantic similarity threshold this value is used to make sure weather the matching keyword is confident enogough to say similar

if (keywordDistances[winnerKeywordId] < SEMANTIC\_DISTANCE\_THRESHOLD):

return winnerKeyword.getLabel()

return NO\_KEYWORD\_MATCHING\_STRING

# getting the whole list of keyword suggesions

def getMatchingKeywordSuggesions(self, phrase):

encording = encodeSentence(phrase)

# getting the winiing centroid

centroidDistances = []

for centroid in self.centroids:

centroidVector = centroid.getDimenVector()

distance = spatial.distance.cosine(encording, centroidVector)

centroidDistances.append(distance)

winnerCentroidId = indexOfMinimum(centroidDistances)

winnerCentroid = self.centroids[winnerCentroidId]

# create an array for store matching keyword suggesions

matchingKeywordSuggesions = []

# getting the most simillar keyword item from the centroid

keywordDistances = []

for keywordItem in winnerCentroid.items:

itemVector = keywordItem.getVector()

distance = spatial.distance.cosine(encording, itemVector)

# adding keywords for suggesions if there exist keywords that below the semantic distance threshold

if (distance < SEMANTIC\_DISTANCE\_THRESHOLD):

matchingKeywordSuggesions.append(keywordItem.getLabel())

keywordDistances.append(distance)

return matchingKeywordSuggesions

# method to evaluate the model with the test dataset

def evaluate(self, x, y):

correctGuesses = 0;

TotalNumber = len(x)

print("%-60s %-60s %-60s" %("Text Phrase","Model Guesed Keyword","Keyword"))

for i in range(0, len(x)):

textPhrase = x[i]

trueKeyword = y[i]

guesedKeyword = self.getMatchingKeyword(textPhrase)

if (guesedKeyword == trueKeyword):

correctGuesses = correctGuesses +1

print("%-60s %-60s %-60s" %(textPhrase, guesedKeyword, trueKeyword))

accuracy = (correctGuesses/TotalNumber)\*100

print("Accuracy ",accuracy,"%")

def evaluateWithMultipleKeywordSuggestions(self, x, y):

correctGuesses = 0;

TotalNumber = len(x)

print("%-60s %-60s %-60s" %("Text Phrase","Model Guesed Keyword","Keyword"))

for i in range(0, len(x)):

textPhrase = x[i]

trueKeyword = y[i]

guesedKeywords = self.getMatchingKeywordSuggesions(textPhrase)

print("%-30s - %-60s" %("Text Phrase", textPhrase))

print("%-30s - %-60s" %("Expected Keyword", trueKeyword))

print()

print("Guessed Keywords")

for guess in guesedKeywords:

print(guess)

if (guess == trueKeyword):

correctGuesses = correctGuesses + 1

print()

accuracy = (correctGuesses/TotalNumber)\*100

print("Accuracy ",accuracy,"%")

# Centroid class

class Centroid:

# constructor of the centroid class

def \_\_init\_\_(self, dimentionality, id, centroidCordinate = None):

self.id = id

self.dimetionality = dimentionality

# If no inizial cordinates are given the random cordinates will be genarated to the centroid

if centroidCordinate is None :

self.cordinates = np.random.rand(1,self.dimetionality)

# If cordinates are given then set those cordinates as the centroid cordinates

else:

self.cordinates = centroidCordinate

self.items = []

# used to set items manually when loading model from file

def setItems(self,items):

self.items = items

# Adding a item to centroid

def addItem(self, item):

self.items.append(item)

# method to query the dimention vector of a centroid

def getDimenVector(self):

return self.cordinates

# method to adjust the dimentions of the centroid accorting to the items in the centroid

# returns a boolian value weather the centroid is adjusted or not

def adjustCentroidCordinates(self):

# first create a vector for new cordinates inizialize with zeros

numOfItems = len(self.items)

newCordinates = np.zeros(shape=(1,self.dimetionality))

# get the number of items as the divider

devider = np.array(numOfItems)

for item in self.items:

# get the vector of each item

vector = item.getVector()

# divide it by devider to get the value it contribute to mean

meanedVector = vector / devider

# add the mean value to new cordinate vector

newCordinates = np.add(newCordinates,meanedVector)

# check weather the new vector is similar to the already exisiting vector

isArraySimilar = np.allclose(self.cordinates,newCordinates)

print(isArraySimilar)

if (isArraySimilar):

# if similar return adjustment finished

return True

else:

# if not similar assign the new cordinates as the cordinates of the centroid and return adjustment is not finished

self.cordinates = newCordinates

return False

# reset the items to a empty list used in the training process

def resetItems(self):

self.items = []

# Keyword item class keyword items objects are used to hold keyword items with in the model

class KeywordItem:

# constructor of the class

def \_\_init\_\_(self,keyword):

self.lable = keyword

# return the keyword of the object

def getLabel(self):

return self.lable

# return the vector representation of the keyword by encording it by bert model

def getVector(self):

encording = encodeSentence(self.getLabel())

return encording

## A.4 Code snippet for training the K-Means model

from KMeans import \*

from fileManipulation import fimport

# importing the keywords dataset and making the keyword item array

filename = 'keywordset.csv'

keywords = []

df = fimport(filename)

for i in range(0, len(df)):

keywords.append( KeywordItem(df.loc[i,'Keyword']) )

# Training parameters

noOfCentroids = 200

noOfKeywords = len(keywords)

gapBetweenRandomCentroids = noOfKeywords//noOfCentroids

modelName = "200centroidRobertaBaseModel.json"

dimentionality = 768

# Creating a random list of centroids from dataset itself to inizially load kmeans model

centroidVectorList = []

for i in range(0,noOfCentroids):

centroidVectorList.append(keywords[gapBetweenRandomCentroids\*i].getVector())

# Making the kmeans model and training it to keyword set

kmeansModel = KMeans(noOfCentroids,dimentionality,centroidCordinates=centroidVectorList)

kmeansModel.fit(keywords)

# saving the trained model to a json file

kmeansModel.saveModelToAFile(modelName)

## A.5 Code snippet for evaluating the model

from KMeans import \*

from fileManipulation import fimport

# filename = "demo\_testset.csv"

filename = "dataset\_final.csv"

df = fimport(filename)

x = []

y = []

for i in range(0, len(df)):

y.append(df.loc[i,"Keyword"])

x.append(df.loc[i,"Text Phrase"])

# Loading model from saved files

modelName = "200centroidRobertaBaseModel.json"

kMeansModel = loadModelFromFile(modelName)

kMeansModel.evaluate(x,y)

kMeansModel.evaluateWithMultipleKeywordSuggestions(x,y)